Pandas數據分析

今天介紹資料分析近來很紅的 pandas 套件, 作者是 Wes McKinney。Python 會成為一個數據分析的熱門語言, 和 pandas 的出現也有相當的關係。

但是 pandas 雖然功能強, 但有些地方沒那麼直覺, 有時會讓大家以為是個深奧的套件。其實你大約可以把 pandas 想成「Python 的 Excel」, 但是功能更強、更有彈性、也有更多的可能性。

下面介紹個基本上就是把 pandas 當 Excel 學的影片, 相信大家會覺得很親切。
https://youtu.be/9d5-Ti6onew (https://youtu.be/9d5-Ti6onew)

In [121]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

1 開始使用 pandas

首先我們來讀入一個 CSV 檔, 這裡有個「假的」學測成績, 叫 `grades.csv` 我們來練習一下。

In [122]:
```
df = pd.read_csv('data\grades.csv')
df
```

Out[122]:

	姓名	國文	英文	數學	自然	社會
0	劉俊安	9	10	15	10	13
1	胡玉華	10	10	10	8	9
2	黃淑婷	13	15	8	11	14
3	陳上紫	10	10	8	9	14
4	崔靜成	13	12	14	12	13
...	...	...	...	...	...	...
95	蔡佳燕	9	10	9	13	14
96	吳筱婷	8	10	14	10	15
97	陳家銘	14	9	11	8	12
98	李明威	15	9	8	9	15
99	農揚勇	9	11	12	12	10

100 rows × 6 columns

用 `df` 是標準的叫法 (雖然這名稱我們隨便取也可以), 意思是 Data Frame, 這是 `pandas` 兩大資料結構之一。我們可以把 Data Frame 想成一張表格 (雖然其實可以是很多張表格)。

我們來看看我們 df 的前五筆資料。

In [123]:
```
type(df)
```

Out[123]:
```
pandas.core.frame.DataFrame
```

In [124]:
```
df.head(5)
```

Out[124]:

	姓名	國文	英文	數學	自然	社會
0	劉俊安	9	10	15	10	13
1	胡玉華	10	10	10	8	9
2	黃淑婷	13	15	8	11	14
3	陳上紫	10	10	8	9	14
4	崔靜成	13	12	14	12	13

如果你曾經手動讀入 CSV 檔, 就知道這省了多少事 (雖然我個人還挺喜歡純手動帶進 CSV)。

Excel 檔也可以快速讀入

不只 CSV 檔, 很多資料檔案, 像 Excel 檔都很容易在 `pandas` 完成。使用法是這樣:

```
df2 = pd.read_excel('filename.xls', 'sheetname')
```

其中 sheetname 那裡要放工作表的名稱, 如果是中文的最好改成英文。

2 Pandas 基本資料結構

Pandas 有兩個基本資料結構:

- **DataFrame**: 可以想成一個表格。
- **Series**: 表格的某一列、某一行, 基本上就是我們以前的 list 或 array

一個 DataFrame, 我們有 `index` (列的名稱), `columns` (行的名稱)。

DataFrame

Series

剛剛說 series 大概就是一個 list, 一個 array。其實更精準的說, 其實是一個有 "index" 的 array。

DataFrame 的每一行或每一列其實也都是一個 series。我們來看個例子, 例如所有同學的國文成績, 就是一個 series。

In [125]:
```
df['國文']
```

Out[125]:
```
0      9
1     10
2     13
3     10
4     13
      ..
95     9
96     8
97    14
98    15
99     9
Name: 國文, Length: 100, dtype: int64
```

在 Python 3 中, 我們終於可以和英文同步, 用這種很炫的方式叫出所有國文成績。

In [126]:
```
df.國文
```

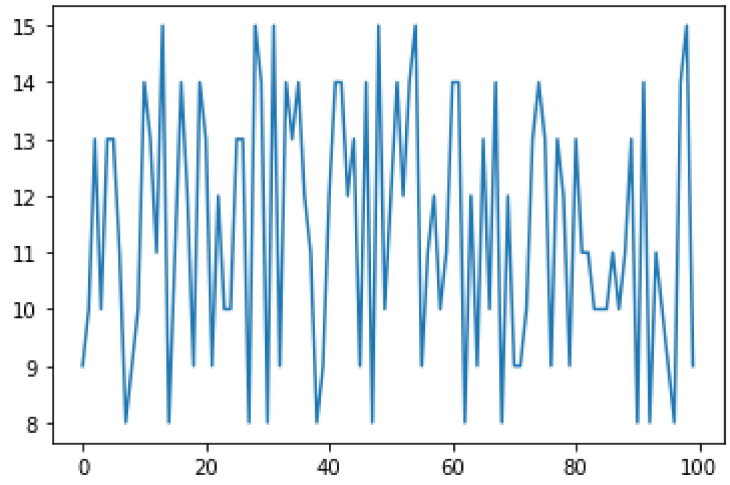
Out[126]:
```
0      9
1     10
2     13
3     10
4     13
      ..
95     9
96     8
97    14
98    15
99     9
Name: 國文, Length: 100, dtype: int64
```

資料畫出來

要畫個圖很容易。

In [127]:
```
df.國文.plot() #plot=畫圖表
```

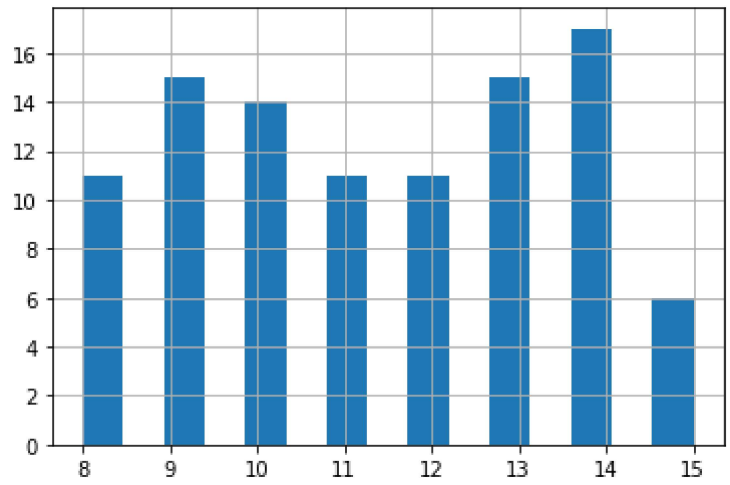
Out[127]: <AxesSubplot:>



當然, 在這個例子中, 其實畫 histogram 圖更有意義一點。

In [128]:
```
df.國文.hist(bins=15) #hist = 畫圖 ,bins = 間隔
```

Out[128]: <AxesSubplot:>



3 一些基本的資料分析

算平均。

In [129]:
```
df.國文.mean()
```

Out[129]: 11.39

算標準差。

In [130]:
```
df.國文.std()
```

Out[130]:
```
2.1968526614594834
```

不如就該算的都幫我們算算...

In [131]:
```
df.describe()
```

Out[131]:

	國文	英文	數學	自然	社會
count	100.000000	100.000000	100.000000	100.00000	100.00000
mean	11.390000	11.380000	11.570000	11.03000	11.83000
std	2.196853	2.273164	2.310516	2.21772	2.48655
min	8.000000	8.000000	8.000000	8.00000	8.00000
25%	9.000000	9.000000	10.000000	9.00000	9.00000
50%	11.000000	11.000000	11.000000	11.00000	12.00000
75%	13.000000	13.000000	14.000000	13.00000	14.00000
max	15.000000	15.000000	15.000000	15.00000	15.00000

有時我們很愛看的相關係數矩陣。

In [132]:
```
df.corr()
```

Out[132]:

	國文	英文	數學	自然	社會
國文	1.000000	0.160158	-0.310899	-0.110236	-0.028421
英文	0.160158	1.000000	0.025656	0.113929	-0.063512
數學	-0.310899	0.025656	1.000000	0.014371	0.041651
自然	-0.110236	0.113929	0.014371	1.000000	-0.156594
社會	-0.028421	-0.063512	0.041651	-0.156594	1.000000

只算兩科間的相關係數當然也可以。

In [133]:
```
df.國文.corr(df.數學) #相關係數
```

Out[133]:
```
-0.3108989822179331
```

4 增加一行

【技巧】

我們增加一行, 加入總級分。

In [134]:
```
df = pd.read_csv('data\grades.csv') #避免重複加總
df["總級分"] = df.sum(axis=1)
df.head(3)
```

Out[134]:

	姓名	國文	英文	數學	自然	社會	總級分
0	劉俊安	9	10	15	10	13	57
1	胡玉華	10	10	10	8	9	47
2	黃淑婷	13	15	8	11	14	61

【技巧】

有計算的當然也可以的。

In [135]:
```
df = pd.read_csv('data\grades.csv') #避免重複加總
df["總級分"] = df.sum(axis=1)
df["加權"] = df.國文 * 0.4 + df.英文 * 0.6 + df.數學 * 0.8
df.head(3)
```

Out[135]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
0	劉俊安	9	10	15	10	13	57	21.6
1	胡玉華	10	10	10	8	9	47	18.0
2	黃淑婷	13	15	8	11	14	61	20.6

5 排序和 index 重設

【重點】排序的方法

我們依總級分來排序。

In [136]:
```
df = df.sort_values(by='總級分',ascending=False)
df.head(3)
```

Out[136]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
80	施雅鈴	13	15	12	13	13	66	23.8
12	李正偉	11	15	11	14	15	66	22.2
54	陳怡潔	15	15	9	15	11	65	22.2

【重點】排序的方法

加權分最高, 同分才看總級分

In [137]:
```
df2 = df.sort_values(by=['加權','總級分'],ascending=False)
df2.head(10)
```

Out[137]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
73	吳志遠	13	15	15	8	8	59	26.2
57	胡淳茜	12	15	14	13	11	65	25.0
36	詹威德	12	13	15	10	14	64	24.6
83	李士賢	10	14	15	13	13	65	24.4
25	蔡亦瑄	13	13	14	13	12	65	24.2
44	童心怡	13	13	14	8	13	61	24.2
48	陳怡婷	15	14	12	9	15	65	24.0
65	謝雅琳	13	14	13	12	9	61	24.0
80	施雅鈴	13	15	12	13	13	66	23.8
37	曾怡君	11	12	15	13	14	65	23.6

【重點】重設 index

In [138]:
```
df2.index = range(1,101) #重新修改index ,顯示1~100
df2
```

Out[138]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
1	吳志遠	13	15	15	8	8	59	26.2
2	胡淳茜	12	15	14	13	11	65	25.0
3	詹威德	12	13	15	10	14	64	24.6
4	李士賢	10	14	15	13	13	65	24.4
5	蔡亦瑄	13	13	14	13	12	65	24.2
...	...	...	...	...	...	...	...	...
96	林金鳳	8	9	10	10	8	45	16.6
97	陳上紫	10	10	8	9	14	51	16.4
98	胡凱琳	10	8	9	9	15	51	16.0
99	梁慧君	9	8	8	8	12	45	14.8
100	吳文伯	8	8	8	15	12	51	14.4

100 rows × 8 columns

6 篩出我們要的資料

基本上和 NumPy 的 array 篩法很像。

【重點】

找出數學滿級分同學。

In [139]:
```
df2[df2.數學==15]
```

Out[139]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
1	吳志遠	13	15	15	8	8	59	26.2
3	詹威德	12	13	15	10	14	64	24.6
4	李士賢	10	14	15	13	13	65	24.4
10	曾怡君	11	12	15	13	14	65	23.6
13	葉儀依	9	13	15	8	14	59	23.4
15	陳竹伯	10	12	15	10	14	61	23.2
17	林哲法	14	9	15	10	14	62	23.0
20	周育霖	9	12	15	13	12	61	22.8
23	張雅彬	10	11	15	12	8	56	22.6
25	芮秋辛	11	10	15	10	8	54	22.4
30	胡勝傑	8	11	15	10	15	59	21.8
37	劉俊安	9	10	15	10	13	57	21.6
57	段冠廷	9	8	15	12	15	59	20.4

【重點】

找出數學和英文都滿級分的同學。要注意 `and` 要用 `&`，`or` 要用 `|`。每個條件一定要加弧號。

In [140]:
```
df3 = df2[(df2.數學 == 15) & (df2.英文 == 15)]
df3
```

Out[140]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
1	吳志遠	13	15	15	8	8	59	26.2

7 刪除一行或一列

【重點】刪掉一行

我們來刪掉總級分的那行。

In [141]:
```
df2.drop('總級分',axis=1)
df2
```

Out[141]:

	姓名	國文	英文	數學	自然	社會	總級分	加權
1	吳志遠	13	15	15	8	8	59	26.2
2	胡淳茜	12	15	14	13	11	65	25.0
3	詹威德	12	13	15	10	14	64	24.6
4	李士賢	10	14	15	13	13	65	24.4
5	蔡亦瑄	13	13	14	13	12	65	24.2
...	...	...	...	...	...	...	...	...
96	林金鳳	8	9	10	10	8	45	16.6
97	陳上紫	10	10	8	9	14	51	16.4
98	胡凱琳	10	8	9	9	15	51	16.0
99	梁慧君	9	8	8	8	12	45	14.8
100	吳文伯	8	8	8	15	12	51	14.4

100 rows × 8 columns

【重點】改變原有的 DataFrame

我們會發現 `pandas` 很多動作都沒有更改原有的 DataFrame, 真的要改要加入

```
inplace=True
```

In [142]:
```
df2.drop('總級分',axis=1,inplace=True)
df2
```

Out[142]:

	姓名	國文	英文	數學	自然	社會	加權
1	吳志遠	13	15	15	8	8	26.2
2	胡淳茜	12	15	14	13	11	25.0
3	詹威德	12	13	15	10	14	24.6
4	李士賢	10	14	15	13	13	24.4
5	蔡亦瑄	13	13	14	13	12	24.2
...	...	...	...	...	...	...	...
96	林金鳳	8	9	10	10	8	16.6
97	陳上紫	10	10	8	9	14	16.4
98	胡凱琳	10	8	9	9	15	16.0
99	梁慧君	9	8	8	8	12	14.8
100	吳文伯	8	8	8	15	12	14.4

100 rows × 7 columns

【重點】刪掉一列

刪掉列就是指定要刪去的 index。

In [143]:
```
df2.drop(5).head(10)
```

Out[143]:

	姓名	國文	英文	數學	自然	社會	加權
1	吳志遠	13	15	15	8	8	26.2
2	胡淳茜	12	15	14	13	11	25.0
3	詹威德	12	13	15	10	14	24.6
4	李士賢	10	14	15	13	13	24.4
6	童心怡	13	13	14	8	13	24.2
7	陳怡婷	15	14	12	9	15	24.0
8	謝雅琳	13	14	13	12	9	24.0
9	施雅鈴	13	15	12	13	13	23.8
10	曾怡君	11	12	15	13	14	23.6
11	崔靜成	13	12	14	12	13	23.6

【重點】刪掉一列

通常刪掉符合條件的比較合理 (注意是找到要刪掉的部份, 再找出相對的 index)。

In [144]:
```
df2.drop(df2[df2.姓名=="吳志遠"].index)
```

Out[144]:

	姓名	國文	英文	數學	自然	社會	加權
2	胡淳茜	12	15	14	13	11	25.0
3	詹威德	12	13	15	10	14	24.6
4	李士賢	10	14	15	13	13	24.4
5	蔡亦瑄	13	13	14	13	12	24.2
6	童心怡	13	13	14	8	13	24.2
...	...	...	...	...	...	...	...
96	林金鳳	8	9	10	10	8	16.6
97	陳上紫	10	10	8	9	14	16.4
98	胡凱琳	10	8	9	9	15	16.0
99	梁慧君	9	8	8	8	12	14.8
100	吳文伯	8	8	8	15	12	14.4

99 rows × 7 columns

8 真實股價資料

有個從 `Pandas` 獨立出來的套件叫 `pandas-datareader` , 幾經波折, 先是 Yahoo! 的財務資料不能用, 後來又是 Google 的資料不能用, 不過至少現在看來 Yahoo! 還可以使用。

安裝 `pandas-datareader` 就標準 `conda` 安裝:

```
conda install pandas-datareader
```

如果裝過, 但很久沒更新就用:

```
conda update pandas-datareader
```

【例子】 分析 Apple 股價

In [145]:
```
import pandas_datareader as pdr
#df = pdr.get_data_yahoo('AAPL')
```

In [146]:
```
# 為防止網路有問題, 我們把這個檔案以 aapl.csv 存起來, 可以這樣讀入。
df = pd.read_csv('data/aapl.csv', index_col="Date")
```

In [147]:
```
df
```

Out[147]:

Date	Open	High	Low	Close	Adj Close	Volume
2009-12-31	30.447144	30.478571	30.080000	30.104286	26.986492	88102700
2010-01-04	30.490000	30.642857	30.340000	30.572857	27.406532	123432400
2010-01-05	30.657143	30.798571	30.464285	30.625713	27.453915	150476200
2010-01-06	30.625713	30.747143	30.107143	30.138571	27.017223	138040000
2010-01-07	30.250000	30.285715	29.864286	30.082857	26.967278	119282800
...	...	...	...	...	...	...
2017-12-04	172.479996	172.619995	169.630005	169.800003	169.800003	32542400
2017-12-05	169.059998	171.520004	168.399994	169.639999	169.639999	27350200
2017-12-06	167.500000	170.199997	166.460007	169.009995	169.009995	28560000
2017-12-07	169.029999	170.440002	168.910004	169.320007	169.320007	25673300
2017-12-08	170.490005	171.000000	168.820007	169.369995	169.369995	23173700

2000 rows × 6 columns

只要最後 300 個交易日!

In [148]:
```
df = df[-300:]
df
```

Out[148]:

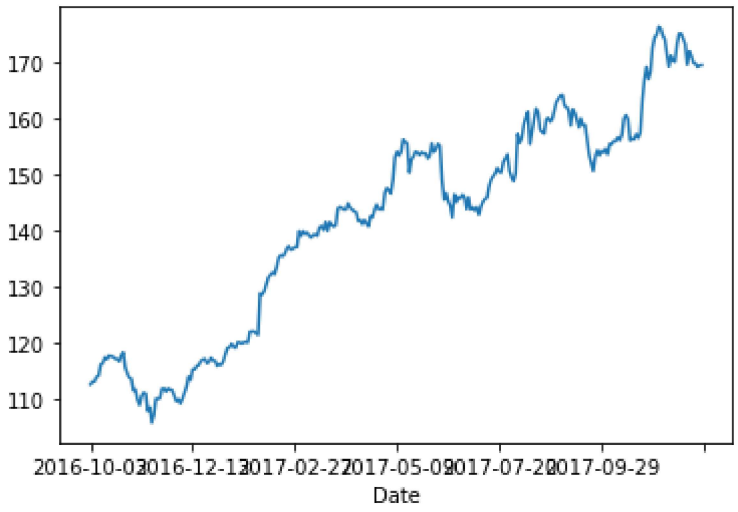
Date	Open	High	Low	Close	Adj Close	Volume
2016-10-03	112.709999	113.050003	112.279999	112.519997	110.173546	21701800
2016-10-04	113.059998	114.309998	112.629997	113.000000	110.643539	29736800
2016-10-05	113.400002	113.660004	112.690002	113.050003	110.692505	21453100
2016-10-06	113.699997	114.339996	113.129997	113.889999	111.514984	28779300
2016-10-07	114.309998	114.559998	113.510002	114.059998	111.681435	24358400
...	...	...	...	...	...	...
2017-12-04	172.479996	172.619995	169.630005	169.800003	169.800003	32542400
2017-12-05	169.059998	171.520004	168.399994	169.639999	169.639999	27350200
2017-12-06	167.500000	170.199997	166.460007	169.009995	169.009995	28560000
2017-12-07	169.029999	170.440002	168.910004	169.320007	169.320007	25673300
2017-12-08	170.490005	171.000000	168.820007	169.369995	169.369995	23173700

300 rows × 6 columns

20 日的移動平均

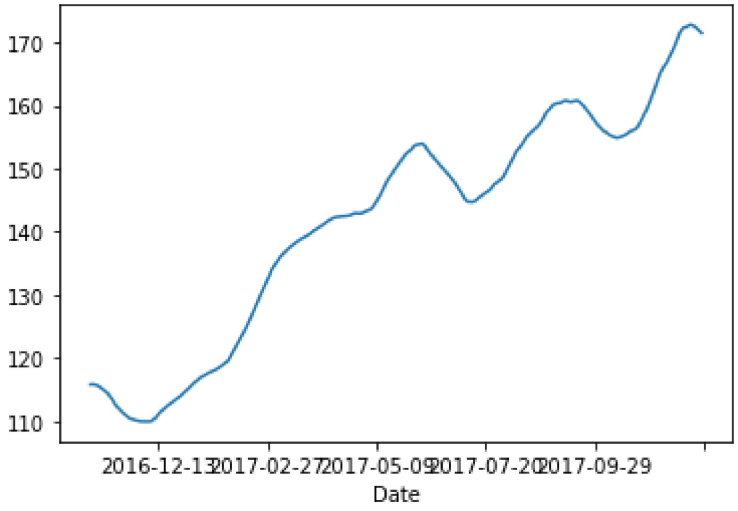
In [149]:
```
df.Close.plot()
df.Close.rolling(20).mean()
```

Out[149]:
```
Date
2016-10-03           NaN
2016-10-04           NaN
2016-10-05           NaN
2016-10-06           NaN
2016-10-07           NaN
                    ...
2017-12-04    172.694001
2017-12-05    172.463501
2017-12-06    172.173501
2017-12-07    171.827501
2017-12-08    171.502000
Name: Close, Length: 300, dtype: float64
```



In [150]:
```
df.Close.rolling(20).mean().plot()
```

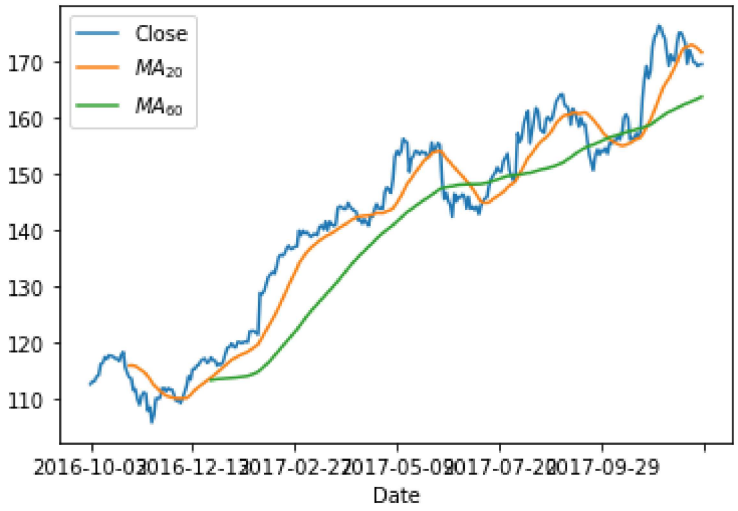
Out[150]:
```
<AxesSubplot:xlabel='Date'>
```



20 日和 60 日的移動平均

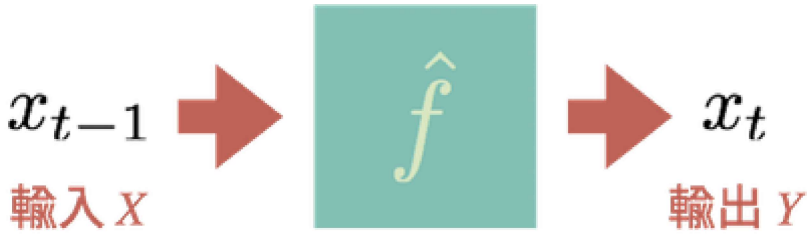
In [151]:
```
df.Close.plot(legend=True)
df.Close.rolling(20).mean().plot(label="$MA_{20}$",legend=True)
df.Close.rolling(60).mean().plot(label="$MA_{60}$",legend=True)
```

Out[151]: <AxesSubplot:xlabel='Date'>



準備做預測

我們用個非常天真的模型...



網路上說這是線性的 (可能嗎)!

In [152]:
```
close = df.Close.values
len(close)
```

Out[152]: 300

In [153]:
```
# 0~299
x = close[:-1]
len(x)
```

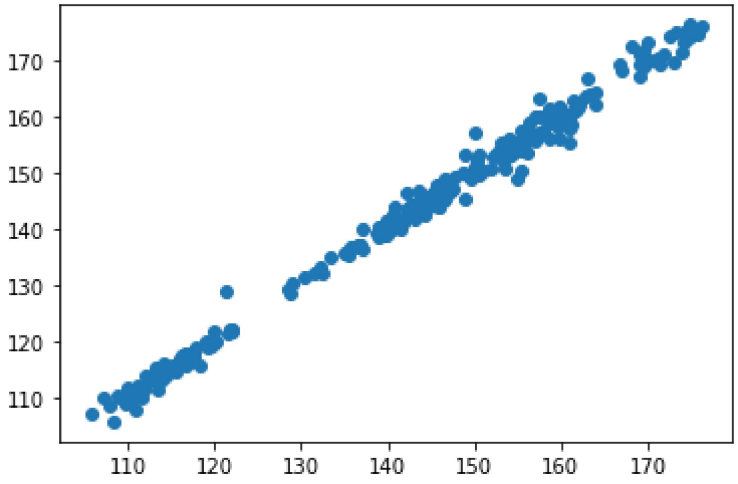
Out[153]: 299

In [154]:
```
#1~300
y = close[1:]
len(y)
```

Out[154]: 299

In [155]:
```
plt.scatter(x,y)
```

Out[155]: <matplotlib.collections.PathCollection at 0x144691c69a0>



哦, 真的有點像線性的, 我們之後用線性迴歸試試看。

9 手工打造一個 DataFrame*

有時我們用手工打造一個簡單的 DataFrame, 可以更理解整個結構。其實很容易, 一個 DataFrame 基本上就包含兩個主要部份:

- 資料本身: 通常一個二維陣列 (矩陣)
- 行、列的名稱

我們來個簡單的小例子。

In [156]:
```
mydata = np.random.randn(5,3)
```

In [157]:
```
mydata
```

Out[157]:
```
array([[ 2.90773362,  1.35515657,  0.33950247],
       [ 0.66018936, -0.21213625, -0.16246924],
       [ 0.04015649,  1.29632956, -1.8464767 ],
       [ 0.51481921, -0.10614608,  0.99842651],
       [ 0.82513091, -0.64932178, -0.96634225]])
```

把行列的名字放進去, 就成一個 DataFrame。我們列的部份先讓 Python 自己產生。

In [158]:
```
df2 = pd.DataFrame(mydata, columns=list("ABC"))
```

In [159]:
```
df2
```

Out[159]:

	A	B	C
0	2.907734	1.355157	0.339502
1	0.660189	-0.212136	-0.162469
2	0.040156	1.296330	-1.846477
3	0.514819	-0.106146	0.998427
4	0.825131	-0.649322	-0.966342

兩個表格上下貼起來

我們再來生一個 DataFrame, 再「貼」起來。

In [160]:
```
df3 = pd.DataFrame(np.random.randn(3,3), columns=list("ABC"))
```

In [161]:
```
df3
```

Out[161]:

	A	B	C
0	0.434598	1.385915	1.257292
1	0.160296	0.399540	-0.632879
2	0.312980	0.261185	1.239465

In [162]:
```
df4=pd.concat([df2,df3],axis=0) #axis 0 = 上下 ,1 = 左右
```

In [163]:
```
df4
```

Out[163]:

	A	B	C
0	2.907734	1.355157	0.339502
1	0.660189	-0.212136	-0.162469
2	0.040156	1.296330	-1.846477
3	0.514819	-0.106146	0.998427
4	0.825131	-0.649322	-0.966342
0	0.434598	1.385915	1.257292
1	0.160296	0.399540	-0.632879
2	0.312980	0.261185	1.239465

前面我們弄得亂七八糟的 index 重設一下。

In [164]:
```
df4.index = range(len(df4))
```

In [165]:
```
df4
```

Out[165]:

	A	B	C
0	2.907734	1.355157	0.339502
1	0.660189	-0.212136	-0.162469
2	0.040156	1.296330	-1.846477
3	0.514819	-0.106146	0.998427
4	0.825131	-0.649322	-0.966342
5	0.434598	1.385915	1.257292
6	0.160296	0.399540	-0.632879
7	0.312980	0.261185	1.239465

橫向的貼

In [166]:
```
df5=pd.concat([df2,df3],axis=1) #axis 0 = 上下 ,1 = 左右
```

等等, 這大小好像不太對也可以嗎? 答案是可以的!

In [167]:
```
df5
```

Out[167]:

	A	B	C	A	B	C
0	2.907734	1.355157	0.339502	0.434598	1.385915	1.257292
1	0.660189	-0.212136	-0.162469	0.160296	0.399540	-0.632879
2	0.040156	1.296330	-1.846477	0.312980	0.261185	1.239465
3	0.514819	-0.106146	0.998427	NaN	NaN	NaN
4	0.825131	-0.649322	-0.966342	NaN	NaN	NaN

大一點的例子

我們來做前面「假的」學測資料。首先要有「假的」同學名單, 如果有興趣產生很多名字, 可以用這個服務。

中文姓名產生器 (http://www.richyli.com/name/index.asp)

In [168]:
```
df_names = pd.read_csv('data/names.csv',names=['姓名'])
```

In [169]:
```
df_names
```

Out[169]:

	姓名
0	羅雅慧
1	王紹尹
2	黃季坤
3	許正書
4	張鈞蓮
...	...
95	陳君生
96	曾桂甫
97	林欣怡
98	林佳霖
99	胡筱芸

100 rows × 1 columns

In [170]:
```
df_grades = pd.DataFrame(np.random.randint(6,16,(100,5)),
                         columns=['國文','英文','數學','社會','自然'])
```

In [171]:
```
df_grades.to_csv('data/grades2.csv') # 輸出 CSV
```

In [172]:
```
df_grades
```

Out[172]:

	國文	英文	數學	社會	自然
0	11	7	12	6	11
1	11	9	13	9	12
2	10	6	14	9	11
3	8	13	8	11	7
4	12	6	15	7	15
...	...	...	...	...	...
95	7	14	9	12	13
96	10	9	12	9	8
97	9	13	13	6	9
98	14	13	7	12	9
99	6	7	10	10	15

100 rows × 5 columns

In [173]:
```
df_score = pd.concat([df_names,df_grades],axis=1)
```

In [174]:
```
df_score
```

Out[174]:

	姓名	國文	英文	數學	社會	自然
0	羅雅慧	11	7	12	6	11
1	王紹尹	11	9	13	9	12
2	黃季坤	10	6	14	9	11
3	許正書	8	13	8	11	7
4	張鈞蓮	12	6	15	7	15
...	...	...	...	...	...	...
95	陳君生	7	14	9	12	13
96	曾桂甫	10	9	12	9	8
97	林欣怡	9	13	13	6	9
98	林佳霖	14	13	7	12	9
99	胡筱芸	6	7	10	10	15

100 rows × 6 columns